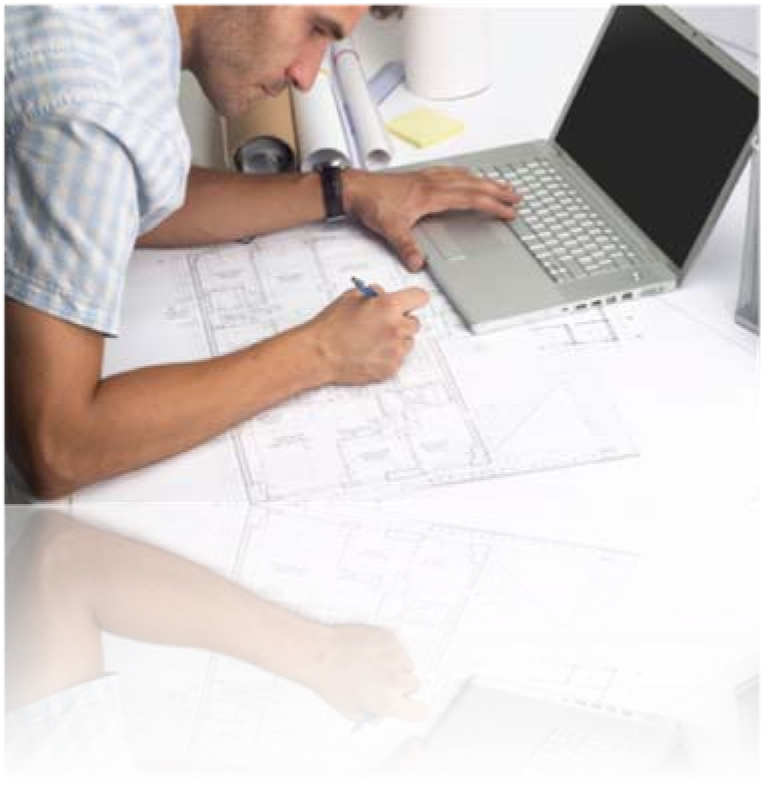


Architectural Patterns that limit Application Scalability

Client + Server Pattern

Server is point of contention

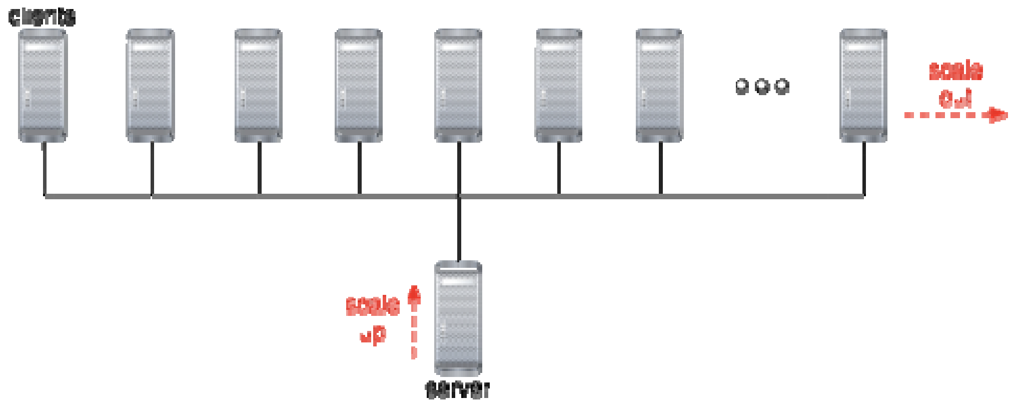
Contention increases Server response time = increased Client latencies

Client scale-out increases contention

Not just Database related. Consider Store-and-Forward messaging systems.

The server may be a “switch”

Lesson: Avoid Single Points of Contention / Bottleneck (SPOB)



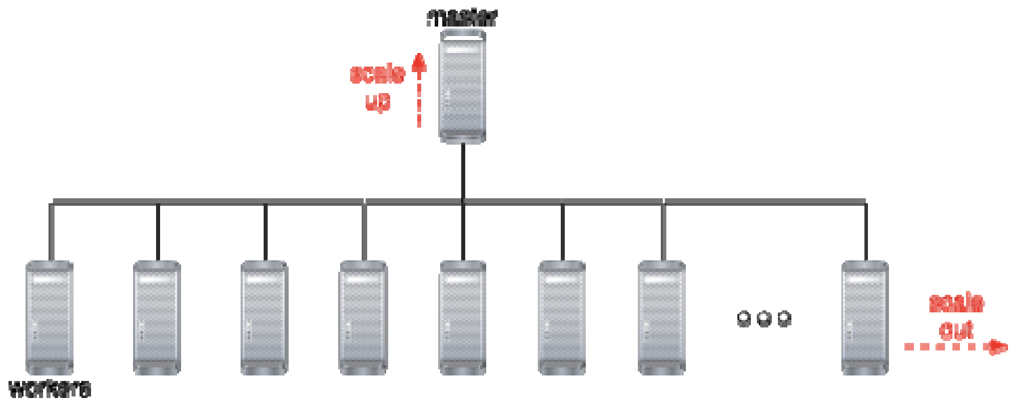
Master + Worker Pattern

Master is point of contention

Contention increases Master response time = increases Worker (and requestor) Latencies

Scale-out increases contention

Lesson: Avoid Single Points of Contention / Bottleneck (SPOB)

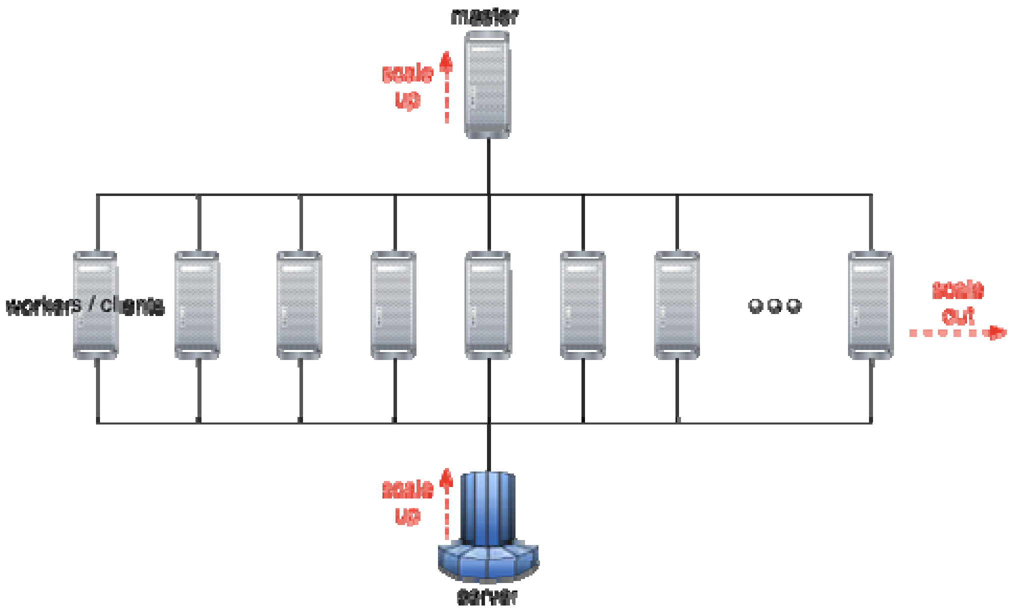


Master + Worker Pattern Continued...

Typically Master + Worker actually is also Client + Server!

Often the driving requirement for “Data Grid” in a “Compute Grid”

Lesson: Avoid patterns with multiple SPOB!



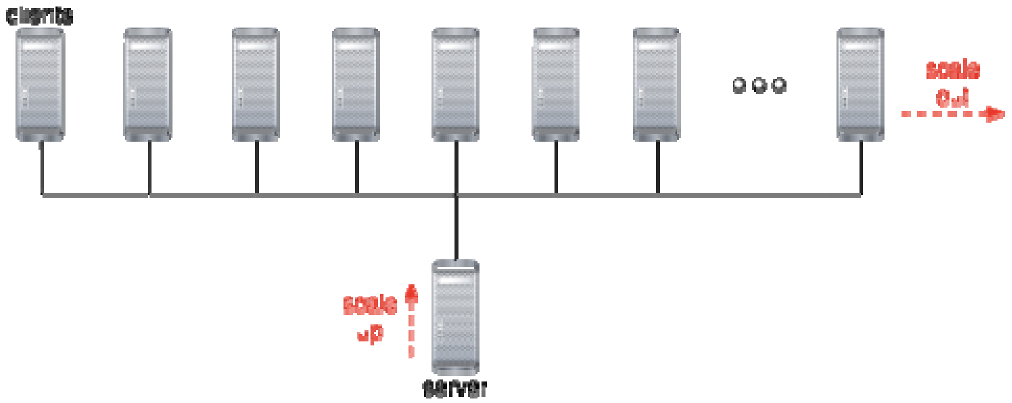
Failure and Recovery (as forms of latency)

System Failure and Recovery introduce latency

Micro Outages (garbage collection, memory management, process management, paging, swapping etc) introduce unexpected latencies

Single Points of Failure may magnify latency effects across a system

Lesson: Avoid Single Points of Failure (SPOF)



Increasing Resilience

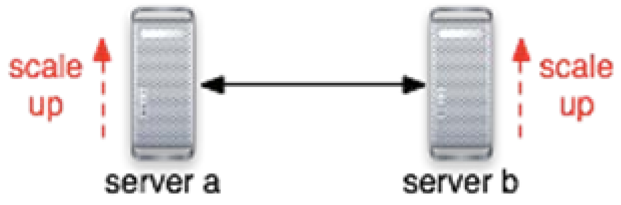
Increasing resilience increases latency

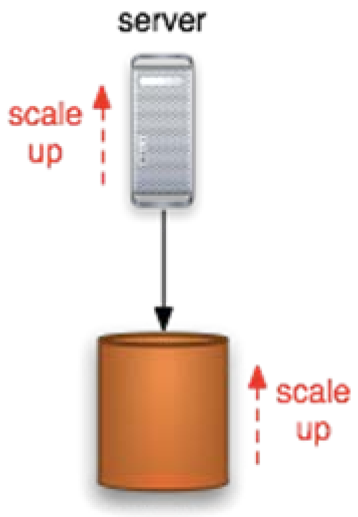
Synchronously maintained resilience typically doubles latencies

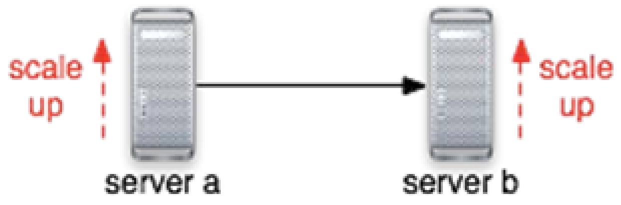
Asynchronously maintained resilience will always introduce data integrity issues

Lesson: Resilience rarely has zero-latency properties

Lesson: Resilience ≠ Persistence







Partition for Parallelism

Partition Data onto separate Masters to provide load-balancing and increase parallelism

Not always easy, especially if access patterns are dynamic and load is uneven

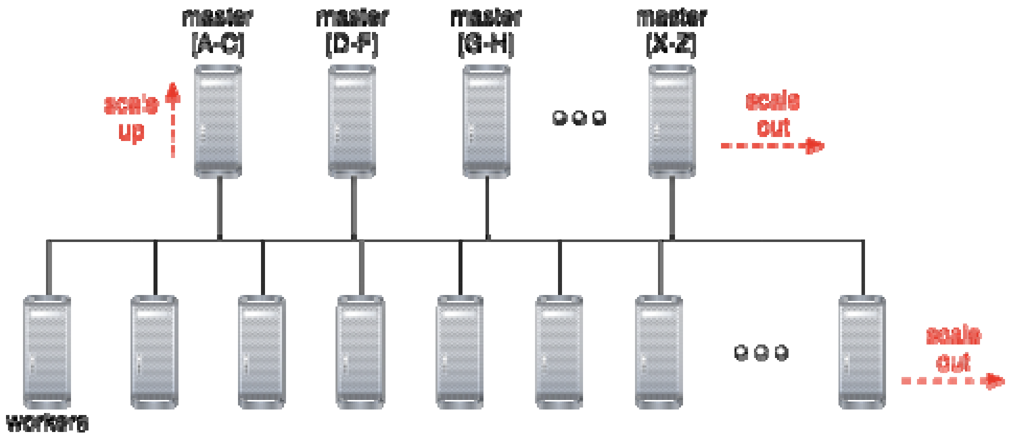
“Joins” become very difficult, but queries work in parallel

Lesson: Hot spots are inevitable

Lesson: Slowest Responders!

Lesson: Partition failure may corrupt state. RAID is a better partitioning strategy

Lesson: Don’t use a “registry” to locate data (Masters)



Common traits...

- Each pattern...
  - Focuses on moving data to servers / compute
  - Has points of contention and failure
- Why do we move terabytes of data across networks to only a few megabytes of code?
  - This is extremely inefficient
  - This limits scalability and performance
- Think about “mineral mining”
  - Do they move all of the earth to the city to be processed?

A solution: Introduce Caching?

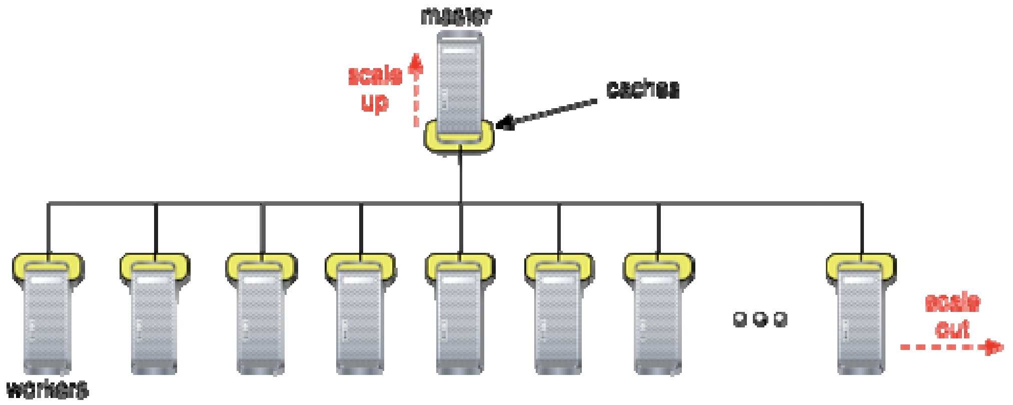
Keep local copies of information to avoid I/O latency.

May offer magnitudes of performance improvement

Lesson: Cache Consistency and Coherency is a challenge as you scale out.

Don't underestimate the effort required to achieve this.

Lesson: "Grid Cache" doesn't mean massive performance improvements – most technologies are implemented using a "faulty" architectural pattern (like ones mentioned previously)



Summary of Lessons

- Avoid Single Points of Contention
- Avoid Single Points of Failure
- Avoid Client + Server
- Avoid Master + Worker
- Active + Active better than Active + Passive
  - Ensure fair utilization of resources
- Resilience increases latency
- Resilience ≠ Persistence
- Resilience = Redundancy
- RAID is a good pattern
- XML is not great
- Interoperability is best achieved at the binary level (hardest, but best)
- Avoid moving data
- Exploit Data Affinity
  - Data + Data and Data + Compute
- Deploy code everywhere
  - It's smaller
  - Dynamic code deployment is dangerous in transactional systems
- Exploit Parallelism
- Partition Data for Parallelism
- Hot Spots are unavoidable
  - Pipeline architectures help significantly
- Use Caching to reduce I/O
- Cache Coherency is not free
- Cache Coherency is essential for Data Integrity
- Understand the underlying implementation of solutions!

Achieving Unlimited Scalability and High Performance means...

1. Doing something completely different architecturally

2. Avoiding patterns that limit scalability or performance

3. Ensuring each vendor supplied architectural component avoids limiting patterns

ORACLE®